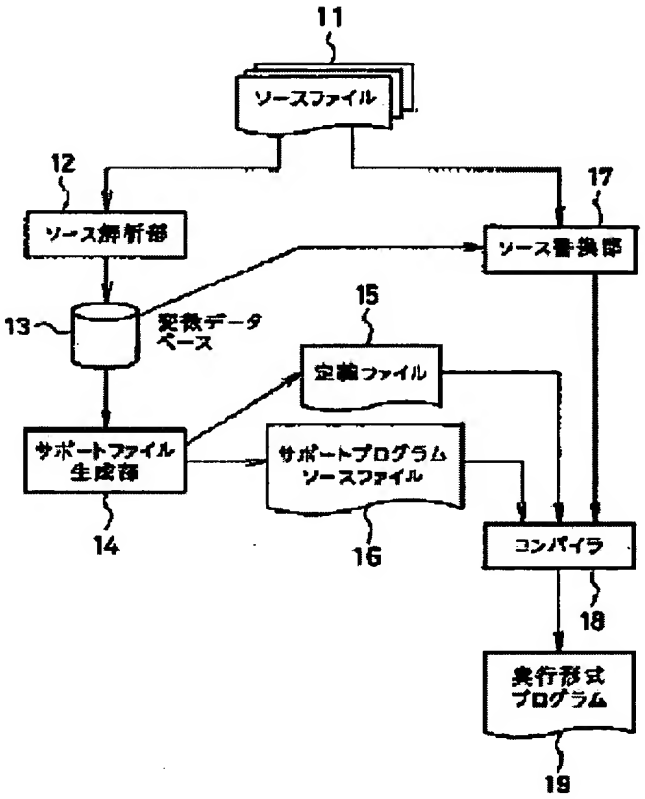
AUTOMATIC RE-ENTRANT METHOD FOR NON-RE-ENTRANT PROGRAM AND SYSTEM THEREFOR

Patent number:	JP11272475
Publication date:	1999-10-08
Inventor:	MURAKAMI TAKUYA
Applicant:	NIPPON ELECTRIC CO
Classification:	
- international:	G06F9/45; G06F9/46
- european:	
Application number:	JP19980092447 19980320
Priority number(s):	JP19980092447 19980320

Report a data error here

Abstract of **JP11272475**

PROBLEM TO BE SOLVED: To provide a method for attaining automatic conversion from a non-re- entrant program into a re-entrant program. SOLUTION: A source file 11 is analyzed by a source analyzing part 12, and the declaring part of a global variable and a static variable is extracted, and registered in a variable data base. A support file generating part 14 generates a definition file in which a definition sentence necessary for a source file after rewriting is described based on the variable data base and a support program source file 16 in which a program such as an initialization function necessary for the initialization of the variable is described. A source rewriting part 17 deletes the definition of the global variable and the static variable registered in the variable data base from the source file, rewrites the variable using part and changes this so that a substitute variable held for each rewiring task can be used, and generates a re-entranced source file. Then, the compile link of this source file with the definition file and the support program is operated, and a re-entrant execution format program is generated.



Data supplied from the ***esp@cenet*** database - Worldwide

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（Ａ）　(11)特許出願公開番号

特開平11－272475

(43)公開日　平成11年(1999)10月8日

(51)Int.Cl.⁶		識別記号	FI		
G06F	9/45		G06F	9/44	322A
	9/46	340		9/46	340F

審査請求　有　請求項の数4　FD　(全 10 頁)

(21)出願番号　特願平10－92447

(22)出願日　平成10年(1998)3月20日

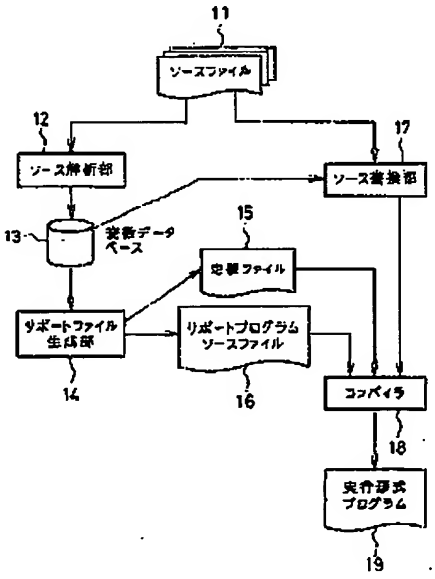
(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目7番1号

(72)発明者　村上　卓弥
東京都港区芝五丁目7番1号　日本電気株式会社内

(74)代理人　弁理士　加藤　朝道

(54)【発明の名称】　非リエントラントプログラムの自動リエントラント化方法及びシステム

(57)【要約】
【課題】非リエントラントプログラムからリエントラントプログラムへの自動変換を行う方法の提供。
【解決手段】ソースファイルをソース解析部で解析し、グローバル変数及びスタティック変数の宣言部を抽出して変数データベースに登録し、サポートファイル生成部で、変数データベースを基に書換え後のソースファイルで必要な定義文を記述した定義ファイルと、変数の初期化に必要な初期化関数などのプログラムを記述したサポートプログラムソースファイルを生成し、ソース書換部でソースファイルから変数データベースに登録されたグローバル変数及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換えタスク毎に保持される代替変数を使用するように変更を行い、リエントラント化されたソースファイルを生成し、これを定義ファイル、サポートプログラムをコンパイルリンクしてリエントラント型の実行形式プログラムを生成する。



1

【特許請求の範囲】

【請求項１】（ａ）非リエントラントプログラムのソースファイルを解析し、グローバル変数．及びスタティック変数の宣言部を抽出してこれを変数データベースに登録し．

（ｂ）前記変数データベースの情報を基にして．書き換え後のソースファイルで必要とされる定義文などを記述した定義ファイルと、変数の初期化に必要な初期化関数及び代替変数アクセス関数などのプログラムを記述したサポートプログラムソースファイルを生成し、

（ｃ）前記ソースファイルから、前記変数データベースに登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換え．タスク毎に保持される代替変数を使用するように変更を行い、リエントラント化されたソースファイルを生成し．

（ｄ）前記リエントラント化されたソースファイルと、前記定義ファイル、及び．前記サポートプログラムをコンパイル及びリンクしてリエントラント型の実行形式プログラムを生成する、

ことを特徴とする非リエントラントプログラムの自動リエントラント化方法。

【請求項２】非リエントラントプログラムのソースファイルを解析し．グローバル変数、及びスタティック変数の宣言部を抽出し、これを変数データベースに登録するソース解析手段と、

前記変数データベースの情報を基にして、書き換え後のソースファイルで必要とされる定義文などを記述した定義ファイルと．変数の初期化に必要な初期化関数及び代替変数アクセス関数などのプログラムを記述したサポートプログラムソースファイルとを生成するサポートファイル生成手段と．

前記ソースファイルから．前記変数データベースに登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換え、タスク毎に保持される代替変数を使用するように変更を行うソース書換手段と、

を備え、

前記ソース書換手段で書換えられたソースファイルと、前記定義ファイル、及び前記サポートプログラムソースファイルをコンパイル及びリンクしてリエントラント型の実行形式プログラムを生成する、ことを特徴とする非リエントラントプログラムの自動リエントラント化システム。

【請求項３】前記ソース書換手段が．前記ソースファイルを走査し、変数が見つかった場合に．該変数が前記変数データベースに登録されているか否かを検索し．登録されている場合には、これが変数の宣言部であるか調べ．変数の宣言部である場合には前記ソースファイルから該宣言部を削除し、一方宣言部でなければ．前記ソースファイルにおける前記変数へのアクセスを、前記サポートプログラムソースファイルに定義されている代替変数アクセス関数へ置き換える、ことを特徴とする請求項２記載の非リエントラントプログラムの自動リエントラント化システム。

2

【請求項４】（ａ）非リエントラントプログラムのソースファイルを解析し、グローバル変数．及びスタティック変数の宣言部を抽出し．これを変数データベースに登録する処理、

（ｂ）前記変数データベースの情報を基にして．書き換え後のソースファイルで必要とされる定義文などを記述した定義ファイルと、変数の初期化に必要な初期化関数及び代替変数アクセス関数などのプログラムを記述したサポートプログラムソースファイルを生成する処理、

（ｃ）前記ソースファイルから、前記変数データベースに登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換え．タスク毎に保持される代替変数を使用するように変更を行う処理．

の上記（ａ）～（ｃ）の各処理をコンピュータで実行させるためのプログラムを記録した記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は．ソフトウェア開発支援システムに関し、特に非リエントラント構造のプログラムをリエントラント構造のプログラムに自動変換する方法及び方式に関する。

【０００２】

【従来の技術】情報処理システムでは．メモリ空間が全タスクで共有される方式をとっているものがある。このようなシステムで、複数のタスクから呼び出される（ｃａｌｌされる）ようなリエントラントなプログラムを作成する場合、以下に説明する２つの方法が知られている。

【０００３】その第１の方法は、このプログラムのコード領域とグローバル変数及びスタティック変数領域を、このプログラムを呼び出すプログラム毎に用意するというものである。しかし、この方法は．メモリの使用効率が悪くなるため．メモリの制約が厳しいシステムでは採用できない場合がある。

【０００４】これを解決するための方法（第２の方法）として、コード領域をこのプログラムを呼び出す全プログラム間で共有させればよい。この場合、このプログラムのコード領域は１つだけあればよいのでメモリの使用効率が良い。

【０００５】しかしながら．コード領域を共有させると．メモリ空間が共有されているシステムでは．グローバル変数及びスタティック変数領域まで共有されてしまうため、プログラムが非リエントラントになってしまい．複数のタスクから呼び出して使用することができない。

【0006】これを回避するためには、プログラム内で使用するグローバル変数、及びスタティック変数の代替となる変数をタスク毎に用意し、タスク毎に自動的に切り替えて使用する機構を導入する必要がある。そして、これを行うには、プログラムのソースファイル自体を修正する必要がある。

【0007】具体的には、プログラムのソースファイルにおいて、グローバル変数とスタティック変数の宣言部の削除と、これらの変数の使用箇所の書き換えが必要である。

【0008】

【発明が解決しようとする課題】従来、この書き換えは、全て人手作業で行う必要があったが、人手作業で行うと、以下に示すような問題が生じることになる。

【0009】（1）プログラムが巨大な場合、人手作業では莫大な労力、作業工数を要する。

【0010】（2）手作業のためミスが発生しやすい。

【0011】（3）ミスを犯した場合、見つけにくいバグの原因となる。

【0012】（4）修正前と修正後の2つのプログラムを保守管理する必要があるため、保守性が低下する。

【0013】このため、この作業を自動化したいという要求があった。この書き換え作業を自動化することができれば、以上の問題は、全て解決することになる。

【0014】したがって、本発明は、上記技術的認識に基づき創案されたものであって、その目的は、非リエントラント型のプログラムからリエントラント型のプログラムへの自動変換を行う方法及びシステムを提供することにある。

【0015】

【課題を解決するための手段】前記目的を達成するため、本発明は、（a）非リエントラントプログラムのソースファイルを解析し、グローバル変数、及びスタティック変数の宣言部を抽出してこれを変数データベースに登録し、（b）前記変数データベースの情報を基にして、書き換え後のソースファイルで必要とされる定義文などを記述した定義ファイルと、変数の初期化に必要な初期化関数及び代替変数アクセス関数などのプログラムを記述したサポートプログラムソースファイルを生成し、（c）前記ソースファイルから、前記変数データベースに登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換え、タスク毎に保持される代替変数を使用するように変更を行い、リエントラント化されたソースファイルを生成し、（d）リエントラント化されたソースファイルと、前記定義ファイル、及びサポートプログラムファイルを参照してコンパイル及びリンクしリエントラント型の実行形式プログラムを生成するようにしたものである。

【0016】

【発明の実施の形態】本発明は、非リエントラント型のプログラムを、メモリ空間が共有されているシステム用のリエントラント型のプログラムに修正するために、自動的にリエントラント化するための手段を提供するものである。より詳細には、本発明は、プログラム内で使用される、リエントラント時に問題になるグローバル変数、及びスタティック変数の削除を、ソースプログラムレベルで行い、このように変更されたソースプログラムに対して、もとのプログラムと同等の動作を行うようにさせるために必要なコード（ソースコード）の追加を自動的に行うものである。

【0017】また、本発明は、プログラムをROM（読み出し専用メモリ）対応プログラムに自動的に修正するための方法も提供する。

【0018】本発明の実施の形態について図面を参照して説明する。図1は、本発明の実施の形態の処理動作を説明するための図である。図1を参照すると、ソースファイル11はリエントラント化の対象となる非リエントラントプログラムのソースコードであり、本発明を適用することにより、このソースファイル11からリエントラント化がなされた、実行形式プログラム19が生成される。本発明は、その好ましい実施の形態において、ソース解析部12、サポートファイル生成部14、ソース解析部17を備える。これらの各部の処理は、コンパイラ18等と同様コンピュータ上で実行されるプログラムで実現される。

【0019】まず、全てのソースファイル11をソース解析部12を用いて解析し、グローバル変数、及びスタティック変数の宣言部を抽出し、これを変数データベース13に登録する。

【0020】次に、この変数データベース13を基にして、サポートファイル生成部14で、定義ファイル15、及びサポートプログラムソースファイル16を生成する。定義ファイル15には、書き換え後のソースファイルで必要とされる定義文などが記述されている。また、サポートプログラムソースファイル16には、変数の初期化に必要な処理などのプログラムが納められる。

【0021】その後、変数データベース13の情報を基にして、ソースファイル11をソース書換部17で書き換える。この際、変数データベース13に登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換えて、タスク毎に保持される代替変数を使用するように変更を行う。

【0022】この時点でリエントラント化されたソースファイルが全て生成されている。このソースファイルを、定義ファイル、サポートプログラムソースファイルとともにコンパイラ18でコンパイルし、リンカでリンクすることでリエントラント化された実行形式プログラム19が得られる。

【0023】

【実施例】上記した本発明の実施の形態についてさらに

【０００６】これを回避するためには、プログラム内で使用するグローバル変数、及びスタティック変数の代替となる変数をタスク毎に用意し、タスク毎に自動的に切り替えて使用する機構を導入する必要がある。そして、これを行うには、プログラムのソースファイル自体を修正する必要がある。

【０００７】具体的には、プログラムのソースファイルにおいて、グローバル変数とスタティック変数の宣言部の削除と、これらの変数の使用箇所の書き換えが必要である。

【０００８】

【発明が解決しようとする課題】従来、この書き換えは、全て人手作業で行う必要があったが、人手作業で行うと、以下に示すような問題が生じることになる。

【０００９】（１）プログラムが巨大な場合、人手作業では莫大な労力、作業工数を要する。

【００１０】（２）手作業のためミスが発生しやすい。

【００１１】（３）ミスを犯した場合、見つけにくいバグの原因となる。

【００１２】（４）修正前と修正後の２つのプログラムを保守管理する必要があるため、保守性が低下する。

【００１３】このため、この作業を自動化したいという要求があった。この書き換え作業を自動化することができれば、以上の問題は、全て解決することになる。

【００１４】したがって、本発明は、上記技術的認識に基づき創案されたものであって、その目的は、非リエントラント型のプログラムからリエントラント型のプログラムへの自動変換を行う方法及びシステムを提供することにある。

【００１５】

【課題を解決するための手段】前記目的を達成するため、本発明は、（ａ）非リエントラントプログラムのソースファイルを解析し、グローバル変数、及びスタティック変数の宣言部を抽出してこれを変数データベースに登録し、（ｂ）前記変数データベースの情報を基にして、書き換え後のソースファイルで必要とされる定義文などを記述した定義ファイルと、変数の初期化に必要な初期化関数及び代替変数アクセス関数などのプログラムを記述したサポートプログラムソースファイルを生成し、（ｃ）前記ソースファイルから、前記変数データベースに登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換え、タスク毎に保持される代替変数を使用するように変更を行い、リエントラント化されたソースファイルを生成し、（ｄ）リエントラント化されたソースファイルと、前記定義ファイル、及びサポートプログラムファイルを参照してコンパイル及びリンクしリエントラント型の実行形式プログラムを生成するようにしたものである。

【００１６】

【発明の実施の形態】本発明は、非リエントラント型のプログラムを、メモリ空間が共有されているシステム用のリエントラント型のプログラムに修正するために、自動的にリエントラント化するための手段を提供するものである。より詳細には、本発明は、プログラム内で使用される、リエントラント時に問題になるグローバル変数、及びスタティック変数の削除を、ソースプログラムレベルで行い、このように変更されたソースプログラムに対して、もとのプログラムと同等の動作を行うようにさせるために必要なコード（ソースコード）の追加を自動的に行うものである。

【００１７】また、本発明は、プログラムをＲＯＭ（読み出し専用メモリ）対応プログラムに自動的に修正するための方法も提供する。

【００１８】本発明の実施の形態について図面を参照して説明する。図１は、本発明の実施の形態の処理動作を説明するための図である。図１を参照すると、ソースファイル１１はリエントラント化の対象となる非リエントラントプログラムのソースコードであり、本発明を適用することにより、このソースファイル１１からリエントラント化がなされた、実行形式プログラム１９が生成される。本発明は、その好ましい実施の形態において、ソース解析部１２、サポートファイル生成部１４、ソース解析部１７を備える。これらの各部の処理は、コンパイラ１８等と同様コンピュータ上で実行されるプログラムで実現される。

【００１９】まず、全てのソースファイル１１をソース解析部１２を用いて解析し、グローバル変数、及びスタティック変数の宣言部を抽出し、これを変数データベース１３に登録する。

【００２０】次に、この変数データベース１３を基にして、サポートファイル生成部１４で、定義ファイル１５、及びサポートプログラムソースファイル１６を生成する。定義ファイル１５には、書き換え後のソースファイルで必要とされる定義文などが記述されている。また、サポートプログラムソースファイル１６には、変数の初期化に必要な処理などのプログラムが納められる。

【００２１】その後、変数データベース１３の情報を基にして、ソースファイル１１をソース書換部１７で書き換える。この際、変数データベース１３に登録されたグローバル変数、及びスタティック変数の定義の削除を行い、変数の使用箇所を書き換えて、タスク毎に保持される代替変数を使用するように変更を行う。

【００２２】この時点でリエントラント化されたソースファイルが全て生成されている。このソースファイルを、定義ファイル、サポートプログラムソースファイルとともにコンパイラ１８でコンパイルし、リンカでリンクすることでリエントラント化された実行形式プログラム１９が得られる。

【００２３】

【実施例】上記した本発明の実施の形態についてさらに

詳細に説明すべく、本発明の実施例について以下に説明する。以下では、Ｃ言語で記述されたソースプログラムを対象とする。

【００２４】図２は、本発明の一実施例の構成を示すブロック図である。図２を参照すると、本実施例は、ソースファイル記憶領域２１と、プリプロセッサ２２と、ソース解析部２３と、変数データベース２４と、サポートファイル生成部２５と、ソース書換部２６と、コンパイラ２７と、リンカ２８と、実行形式プログラムファイル記憶領域２９と、を備えて構成されている。

【００２５】ソースファイル記憶領域２１は、プログラムのソースプログラムファイルを記憶しておく領域である。

【００２６】プリプロセッサ２２は、Ｃプログラム用のプリプロセッサ（前処理系）である。

【００２７】ソース解析部２３は、ソースプログラム中のグローバル変数、及びスタティック変数の定義部を抽出し、この情報を変数データベース２４に登録する。

【００２８】サポートファイル生成部は、変数データベース２４に格納された情報から、ソース書換部２６およびコンパイラ２７で必要になるファイルを生成する。

【００２９】ソース書換部２６は、ソースプログラム中のグローバル変数、及びスタティック変数の宣言部および使用部（プログラム中でグローバル変数、スタティック変数をアクセスする部分）を書き換えたものを出力する。

【００３０】コンパイラ２７は、この出力をコンパイルしてオブジェクトコードを出力し、リンカ２８はコンパイラ２７が生成したオブジェクトコードをリンクして、実行形式プログラムを生成する。リンカ２８が生成する実行形式プログラムファイルは、実行形式プログラムファイル記憶領域２９に保存される。

【００３１】図３は、本発明の一実施例の処理を説明するための図である。図２及び図３を参照して、本発明の一実施例の動作について以下に説明する。図３において、変数データベース３４は図２の変数データベース２４に対応し、プロプロセス３２、ソース解析３３、ソース書換３７等は、図２のプリプロセッサ２２、ソース解析部２３、ソース解析部２６等の処理を表している。

【００３２】まず、ソースファイル３１を、プリプロセッサ２２でプリプロセス（前処理）する（３２）。その後、ソース解析部２３で解析処理を行い（３３）、スタティック変数データベース３４を出力する。

【００３３】図４は、本発明の一実施例におけるソース解析部２３の処理フローを説明するための流れ図である。図４の流れ図を参照して、ソース解析部２３の処理について説明する。

【００３４】初めに、ステップＡ１で、ソースファイル３１を構文解析し、変数定義を探す。

【００３５】次にステップＡ２で、ファイルの終端（ＥＯＦ）に達したかどうか否かを判定し、終端に達したならば処理を終了する。

【００３６】変数が見つかったら、ステップＡ３で、その変数が関数内で定義されたローカル（ｌｏｃａｌ）なものかである否かを調べ、ローカルだった場合、ステップＡ４で、その変数に、さらにstatic（スタティック）キーワードがついているか否かを調べ、付いていなければ、その変数は無視してステップＡ１に戻る。

【００３７】ステップＡ３で変数がローカルでない場合、及びローカルである場合で且つスタティックと定義されている場合にはステップＡ５に進む。

【００３８】次に、ステップＡ５でconst（コンスタント）キーワードが付いているか否かを調べる。constが付いている場合、その変数は定数であって書き換えは不可能であるため、リエントラント化の考慮を行う必要はない。よって、この変数は無視してステップＡ１に戻る。

【００３９】次に、ステップＡ５において変数がconstでない場合、ステップＡ６で、変数の初期化処理があるか否かを調べ、初期化処理があれば、ステップＡ７で、その初期化処理を取り出す。

【００４０】最後に、ステップＡ８で変数データベース３４に登録し、ステップＡ１に戻る。

【００４１】一例として、図６に示したソースプログラムを解析した結果得られる変数データベースの例を図７に示す。

【００４２】データベースには、変数の型名（整数型int、文字型char等）、スコープ（変数がプログラム中で有効となる範囲を表す情報、global、sub()等）、変数名、初期化処理を記録する。

【００４３】再び図２及び図３を参照すると、また、この変数データベース２４から、サポートファイル生成部２５で必要なヘッダファイル３５とサポートプログラムソースファイル３６を生成する。

【００４４】ヘッダファイル３５には、グローバル変数、及びスタティック変数の代替変数をひとまとめにした構造体の定義、およびマクロ、外部関数定義が入る。

【００４５】また、サポートプログラムソースファイル３６には、初期化が必要なグローバル変数及びスタティック変数の初期化処理ルーチンと、代替変数アクセスルーチン処理が入る。

【００４６】図７のデータベースから作成されるヘッダファイル（qsdef.h）の例を図８に、サポートプログラムファイル（qsimt.c）の例を図９に示す。

【００４７】なお、変数名が衝突しないように名前付けを行う必要がある。この例では、例えば、図６に示したソースプログラムの関数sub中の変数cは、図８に示すように、_sub_cという代替の名前を付けている。なお、この例では、ヘッダファイル（qsdef.h）には、グローバル変数（_global_a、_global_b）、及びスタティック

変数（_sub_d）の代替変数をひとまとめにした構造体(struct GSData)の定義、およびマクロ（#define GSDAT(x) (gsdat()->(x))）、外部関数定義（extrern struct GSData *gsdat(void)）が導入される。

【００４８】またサポートプログラムファイル（qsinit.c）では、ヘッダファイル（qsdef.h）をインクルードファイルとし（#include）、初期化処理関数（InitGSData）として、ポインタ変数pで参照される代替変数構造体のうちb（p->_global_b）を２０、d（p->_sub_d）を5に設定している。そして、代替変数構造体アクセス関数（GetGSData）は、グローバル変数及びスタティック変数のタスク毎に固有な代替変数の構造体を返す関数であり、タスク固有のＩＤ（識別）番号から対応する代替変数の構造体GSDataへのポインタpを関数値として返却する。すなわち、この構造体GSDataへのポインタpにはタスクＩＤ値からタスク毎のGSDataのテーブルが探索され、もしpがnullの場合、pを新たに割り当ててテーブルに登録し、初期化関数InitGSData（引数p）を呼び出して初期化処理を行った後に返却する（return p）。

【００４９】再び図２及び図３を参照すると、次に、プリプロセス済みのソースファイルをソース書換部２６を使って書き換える（３７）。

【００５０】図５は、本発明におけるソース書換部２６の処理フローを説明するための流れ図である。図５を参照して、ソース書換部２６の処理について説明する。

【００５１】まず、ステップＢ１、Ｂ２でソースコードの先頭から順に変数定義を全て探す。変数が見つかったら、ステップＢ３、Ｂ４で、それが変数データベース２４に登録されているか否かを調べる。登録されていた場合、ステップＢ５で、それが変数の宣言部であるか否かを調べ、宣言部であれば、ステップＢ６で削除する。

【００５２】宣言部でなければ、ステップＢ７で書き換えを行い、代替変数へのアクセスに切り替える。

【００５３】これは、サポートプログラム中に含まれるアクセスルーチンの呼び出しに置き換えることで行う。

【００５４】このようにして、図６に示したソースプログラムを書き換えた例を図１０に示す。

【００５５】図９のサポートプログラムファイル中には、関数 'GetGSData' が含まれている。この関数は、グローバル変数及びスタティック変数のタスク毎に固有な代替変数のセット（構造体）を返す関数であり、タスク固有のID番号から対応する代替変数のセットを返すようになっている。

【００５６】ソース書換後のプログラムでは、全てこの関数を経由して代替変数にアクセスするようになっている。

【００５７】このため、このプログラムを使用するタスク毎に代替変数が切り替わることになり、プログラムがリエントラント化される。

【００５８】再び図２及び図３を参照すると、この書換後のソースファイルとヘッダファイル３５を用いて再びプリプロセスを行い（３８）、コンパイラ２７を用いてコンパイルする（３９）。

【００５９】また、サポートプログラムソースファイル３６も同様にコンパイルする。得られたオブジェクトファイルをリンカ２８を使ってリンクし（３１０）、最終的にリエントラントな実行形式プログラムが得られる（３１１）。

【００６０】以上、Ｃ言語の場合の例に説明したが、ソース解析部、サポートファイル生成部、ソース書換部を、他言語対応のものに置き換えることにより、所望の言語への適用が可能である。

【００６１】本発明の第二の実施例として、C++言語に適用する場合について説明する。

【００６２】C++言語に適用する場合、以下の２つの方法がある。

１．C++ソースをCに変換し、C言語の場合と同様に処理する。

２．C++ソースをそのまま処理する。

【００６３】第１の方法については、ＡＴ＆Ｔ社製によるcfrontというC++トランスレータを使うことにより、C++をCのソースファイルに変換することができる。C言語のソースファイルに対して、上記した本発明の実施例がそのまま適用される。

【００６４】第２の方法については、ほぼCの場合と同じように処理できるが、C++ソースのｃｌａｓｓ（クラス）内で宣言されているスタティック変数が使用されることがあるので、それを考慮する必要がある。そのため、図２のソース解析部３のアルゴリズムを修正し、ｃｌａｓｓ内スタティック変数も変数データベースに登録するようにする。

【００６５】また、C++ではグローバル及びスタティックなオブジェクトが存在した場合、サポートプログラム内の初期化処理において、このオブジェクトのコンストラクタを呼び出す処理を追加する。

【００６６】さらに、本発明は、プログラムをＲＯＭ対応プログラムに修正する際にも応用することができる。プログラムをＲＯＭ化する場合、初期化が必要なグローバル変数及びスタティック変数をＲＯＭ化することができないため、これを取り除かねばならない。

【００６７】この場合、上記した本発明の実施例の方法をそのまま適用すれば、これらの変数を取り除くことができるため、簡単にＲＯＭ対応を行うことができる。

【００６８】本発明をＣ言語に適用する場合において、図２のソース解析部２３を一部修正し、変数データベース２４には、初期化が必要な変数のみを登録するようにしてもよい。

【００６９】

【発明の効果】以上説明したように、本発明によれば、

下記記載の効果を奏する。

【００７０】本発明の第１の効果は、リエントラント化のための人手による変換作業・労力を完全に不要とすることができ、開発効率を向上し作業工数を特段に削減することができる、ということである。

【００７１】その理由は、本発明においては、リエントラントプログラムへの変換を完全に自動化しており、人手作業を一切必要としないためである。

【００７２】本発明の第２の効果は、プログラムのリエントラント化に伴うバグの発生が生じない、ということである。

【００７３】その理由は、本発明においては、変換にエディット（編集）作業等の、手作業が介入せず、変数名の見落とし、書き換え間違いなどの人為ミスが混入することがないためである。

【００７４】本発明の第３の効果は、保守性を向上する、ということである。

【００７５】その理由は、本発明においては、ソースファイルを自動修正できるため、修正後のソースファイルを保守管理する必要がないためである。

【００７６】本発明の第４の効果は、開発環境および実行環境への依存度が低く、汎用性、可用性が高い、ということである。

【００７７】その理由は、本発明においては、ソースファイルに対して修正を行うのみであるため、特定のコンパイラやＯＳ（オペレーティングシステム）に依存しないためである。

【図面の簡単な説明】

【図１】本発明の実施の形態を説明するための図である。

【図２】本発明の一実施例の構成を示すブロック図である。

【図３】本発明の一実施例の処理の流れを示す図である。

【図４】本発明の一実施例におけるソース解析部の処理フローを示す流れ図である。

【図５】本発明の一実施例におけるソース書換部の処理フローを示す流れ図である。

【図６】本発明の一実施例を説明するための図であり、非リエントラントソースプログラムの一例を示す図である。

【図７】本発明の一実施例を説明するための図であり、変数データベースの一例を示す図である。

【図８】本発明の一実施例を説明するための図であり、サポートファイル生成部で生成されたヘッダファイルの一例を示す図である。

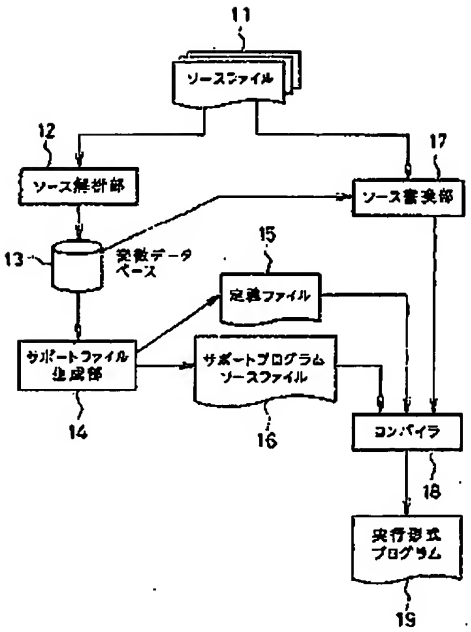
【図９】本発明の一実施例を説明するための図であり、サポートファイル生成部で生成されたサポートプログラムファイルの一例を示す図である。

【図１０】本発明の一実施例を説明するための図であり、リエントラント化されたソースプログラムの一例を示す図である。

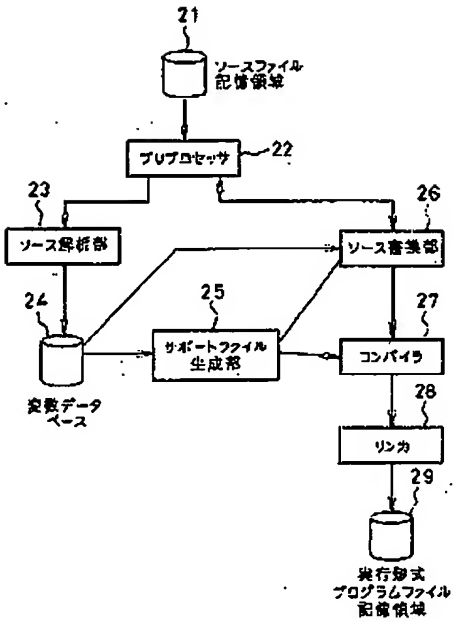
【符号の説明】

１１　ソースファイル
１２　ソース解析部
１３　変数データベース
１４　サポートファイル生成部
１５　定義ファイル
１６　サポートプログラムソースファイル
１７　ソース書換部
１８　コンパイラ
１９　実行形式プログラム
２１　ソースファイル記憶領域
２２　プリプロセッサ
２３　ソース解析部
２４　変数データベース
２５　サポートファイル生成部
２６　ソース書換部
２７　コンパイラ
２８　リンカ
２９　実行形式プログラム記憶領域
３１　ソースファイル
３２　プリプロセス
３３　ソース解析部
３４　変数データベース
３５　ヘッダファイル
３６　サポートプログラムソースファイル
３７　ソース書換
３８　プリプロセス
３９　コンパイル
４０　リンク
４１　実行形式プログラム

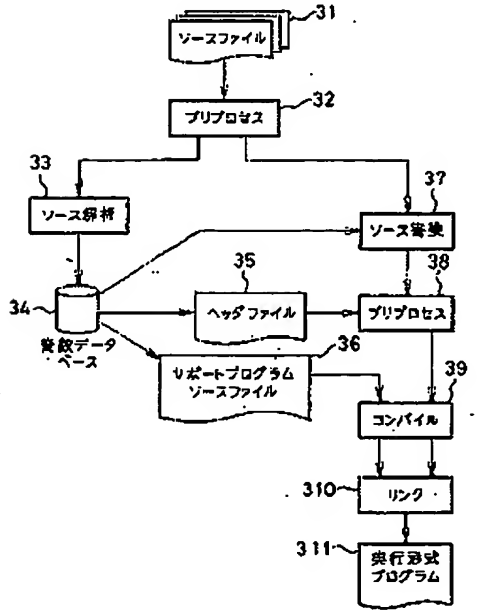
【図1】



【図2】



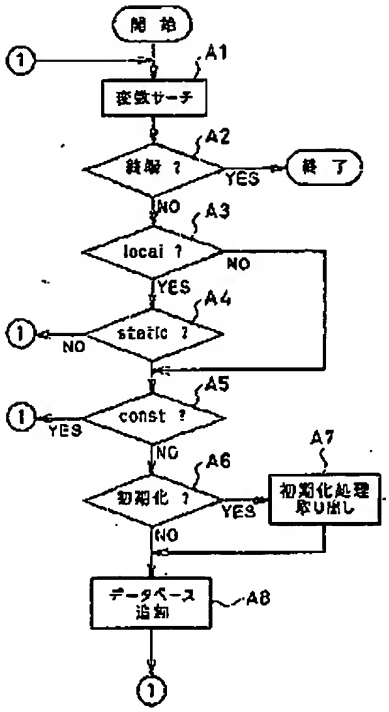
【図3】



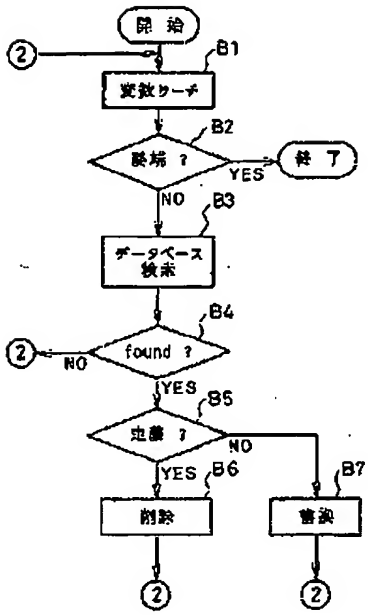
【図6】

```
int   a;
int   b=20;
const int c=100;

void  sub(void)
{
      static  char d=5;
      int e=10;

      a=30;
      d=40;
}
```

【図4】



【図5】



【図7】

型	スコープ	変数名	初期化
int	global	a	-
int	global	b	=20
char	sub()	d	=5

【図8】

```
/* gsdefn */
/* グローバル変数及びスタティック変数の
   代替変数構造体 */
struct GSData {
        int_global_a;
        int_global_b;
        char_sub_d;
};
extern struct GSData *gsdat(void);
#define GSDAT()        (gsdat()->(x))
```

【図9】

```
/*gsinit.c/*
#include "gsdef.h"

/*初期化処理/*
void Init GSData(struct StaticData *p)
{
        p=_global_b=20;
        p=_sub_d=5;
}

/*代替変数構造体取得処理(タスク毎に固有)/*/
struct GSData * Get GSData(void)
{
        struct GSData *p;
        p=/* タスクのID値から、タスク毎のGSDataのテーブルを探す */
        if(p){/*もしテーブルに登録されていなかったら */
                p=/*領域に割り当て、テーブルに登録する */
                InitGSData(p); /*そして、初期化処理を行う */
        }
        return p;
}
```

【図10】

```
#include"gsdefh"

const int c=100;

void sub(void)
{
        int e=10;

        GSDAT(_global_a)=30;
        GSDAT(_sub_d)=40;
}
```

【手続補正書】
【提出日】平成10年11月16日
【手続補正1】
【補正対象書類名】図面
【補正対象項目名】図6
【補正方法】変更
【補正内容】
【図6】

```
int   a;
int   b=20;
const const int c=100;

void sub(void)
{
      static char d=5;
      int e=10;

      a=30;
      d=40;
}
```

*

```
/* gsdefh */
/* グローバル変数及びスタティック変数の
   代替変数構造体 */
struct GSData {
      int _global_a;
      int _global_b;
      char _sub_d;
};
extern struct GSData *gsdat(void);
#define GSDAT(x)      (gsdat()->(x))
```

【手続補正２】
【補正対象書類名】図面
【補正対象項目名】図８
【補正方法】変更
【補正内容】
【図８】

【手続補正３】
【補正対象書類名】図面
【補正対象項目名】図９
【補正方法】変更
【補正内容】
【図９】

```
/*gsinit.c/*
#include "gsdef.h"

/*初期化処理/*
void Init GSData(struct StaticData *p)
{
      p->_global_b=20;
      p->_sub_d=5;
}

/*代替変数構造体取得処理(タスク毎に固有)/*/
struct GSData * Get GSData(void)
{
      struct GSData *p;
      p=/* タスクID値からタスク毎のGSDataのテーブルを探す */
      if(p==/*まだテーブルに登録されていなかったら */
      {
            p=/*新規に割り当て,テーブルに登録する */
            InitGSData(p);/*そして,初期化処理を行う */
      }
      return p;
}
```